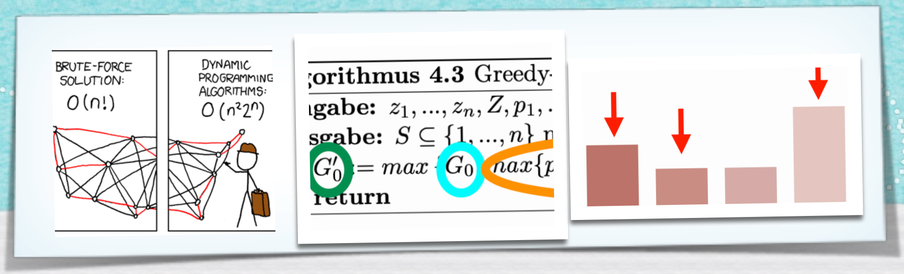


4 Approximation

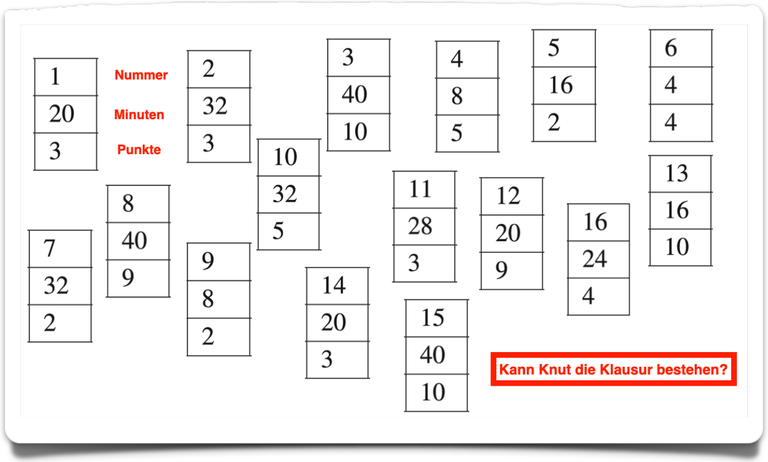
Algorithmen und Datenstrukturen 2
Sommer 2024

Prof. Dr. Sándor Fekete

4.1 Motivation

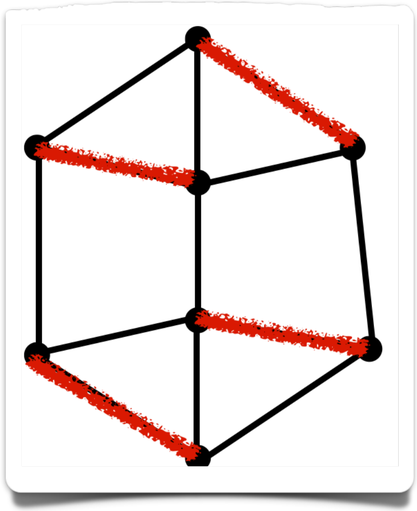
Optimierungsprobleme

0-1 Knapsack



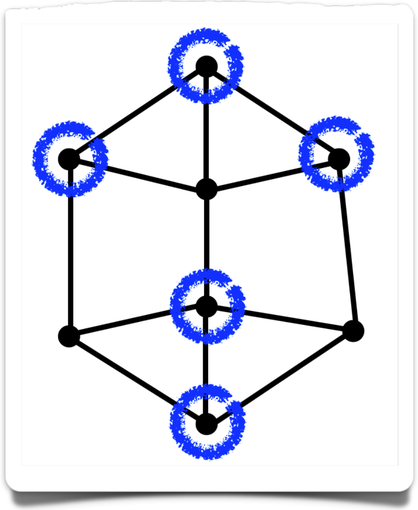
Max

Matching



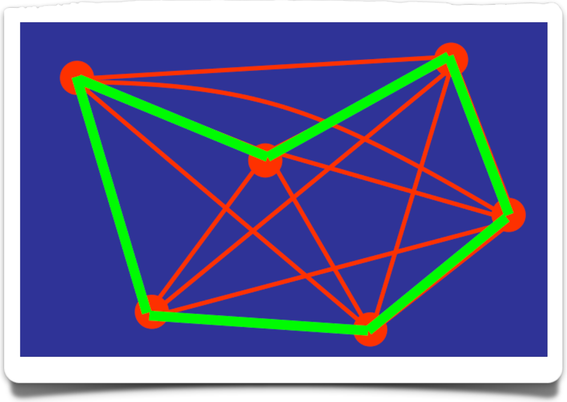
Max

Vertex Cover



Min

Traveling Salesman



Min

Ziel: Finde eine bestmögliche Lösung unter **exponentiell** vielen möglichen!

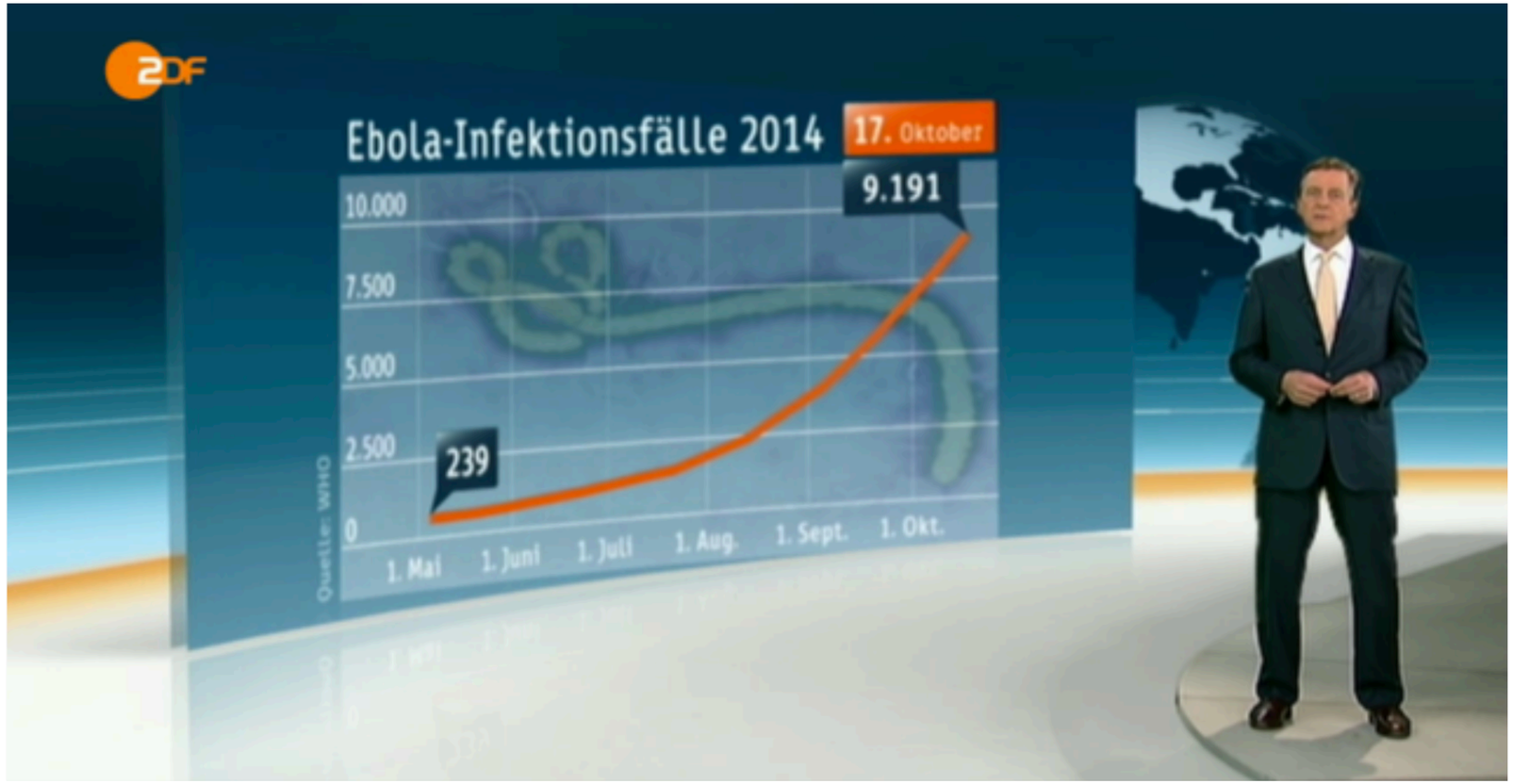
Optimierungsprobleme

0-1

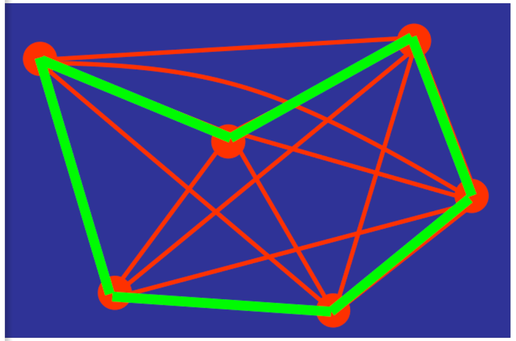
1	Nummer	2
20	Minuten	32
3	Punkte	3

8		10
40		32
9		5

7		9
32		8
2		2



Traveling Salesman



Min

Ziel: Finde eine bestmögliche Lösung unter **exponentiell** vielen möglichen!

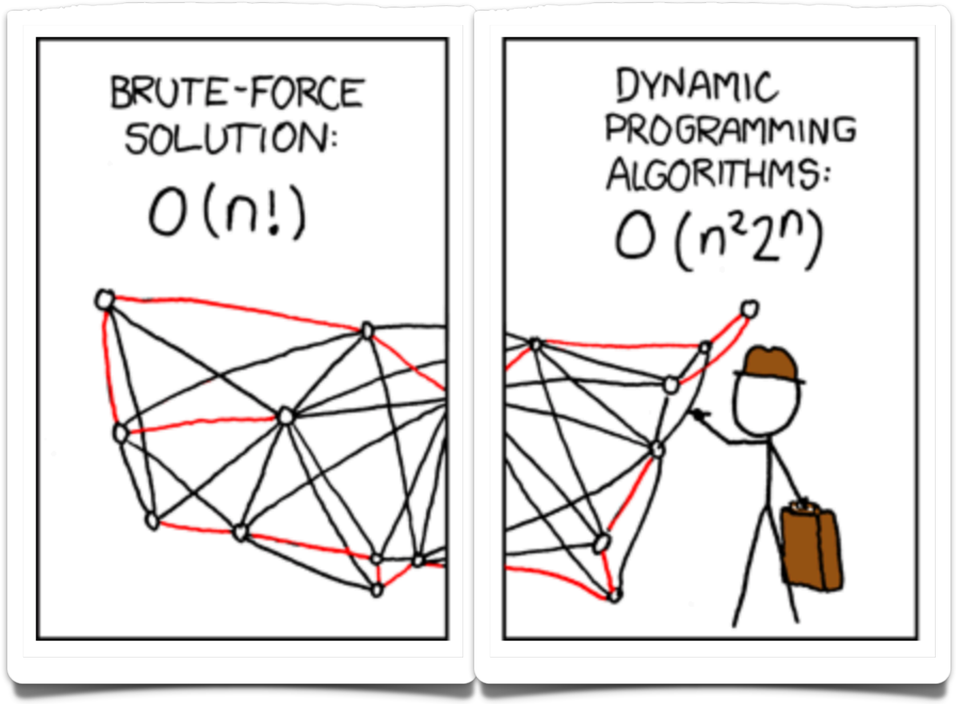
Laufzeit?

0-1 Knapsack

$O(nZ)$

$O(2^n f(n))$

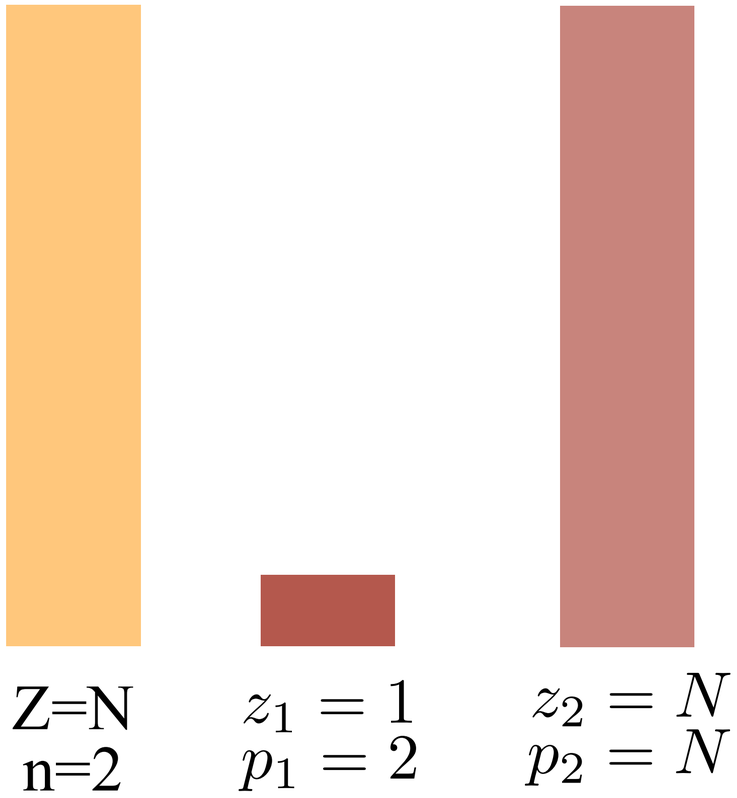
Traveling Salesman



Ziel: Vielleicht nicht die *bestmögliche* Lösung, sondern eine *gute,* aber dafür schneller…?!

4.2 Greedy und Approximation

Beispiel 4.1: Knapsack



Algorithmus 4.2 GREEDY$_0$

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

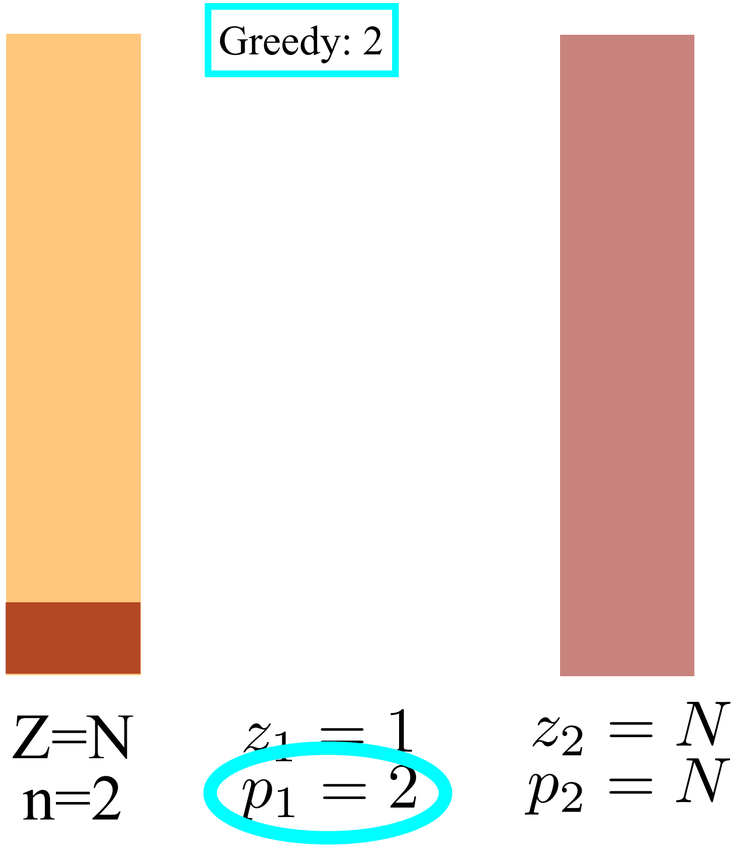
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Maximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$
6: **return**

Beispiel 4.1: Knapsack



Algorithmus 4.2 GREEDY$_0$

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

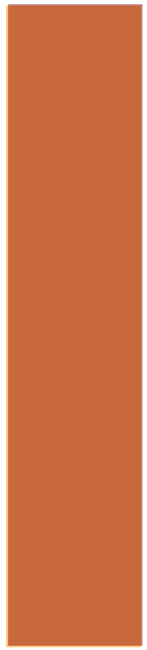
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Maximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$
6: **return**

Beispiel 4.1: Knapsack



Greedy: 2

OPT: N

Z=N
n=2

$z_1 = 1$
$p_1 = 2$

$z_2 = N$
$p_2 = N$

Algorithmus 4.2 GREEDY$_0$

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

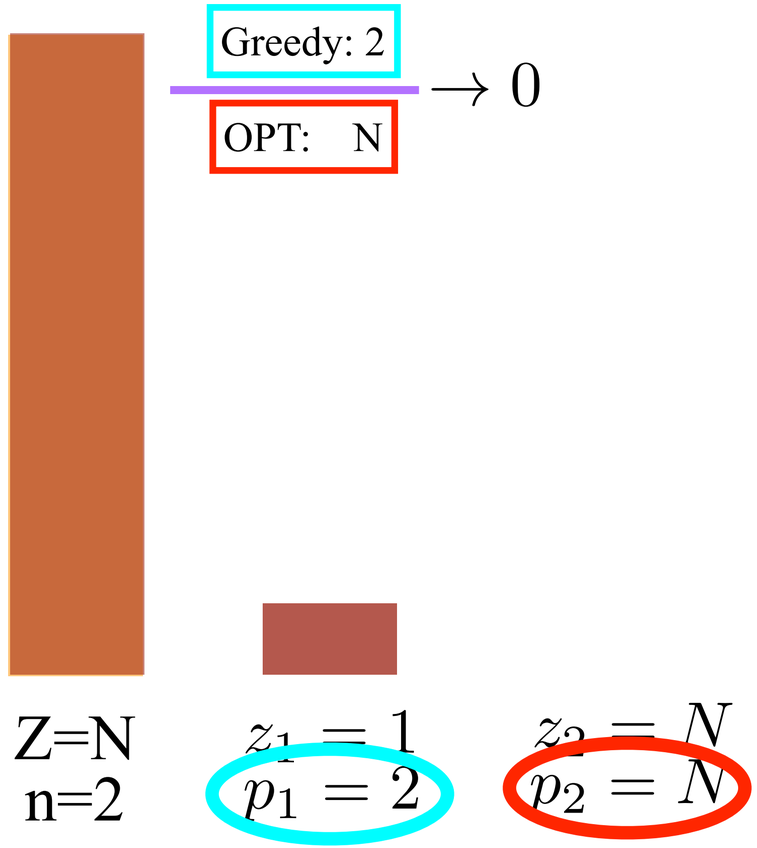
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Maximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$
6: **return**

Beispiel 4.1: Knapsack



Algorithmus 4.2 GREEDY$_0$

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

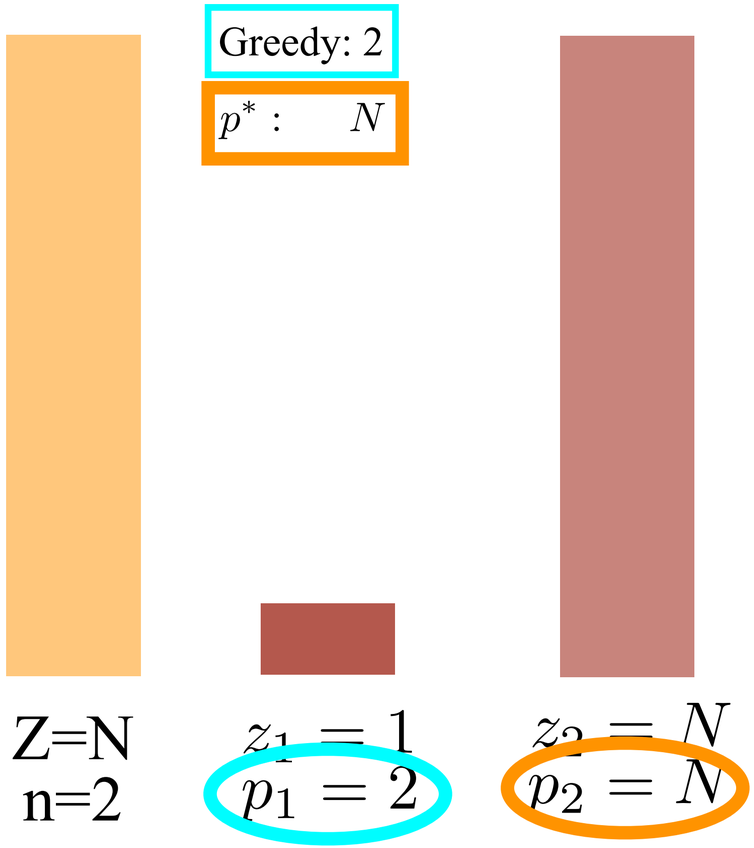
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Maximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$
6: **return**

Verbesserung?!



Algorithmus 4.2 GREEDY$_0$

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

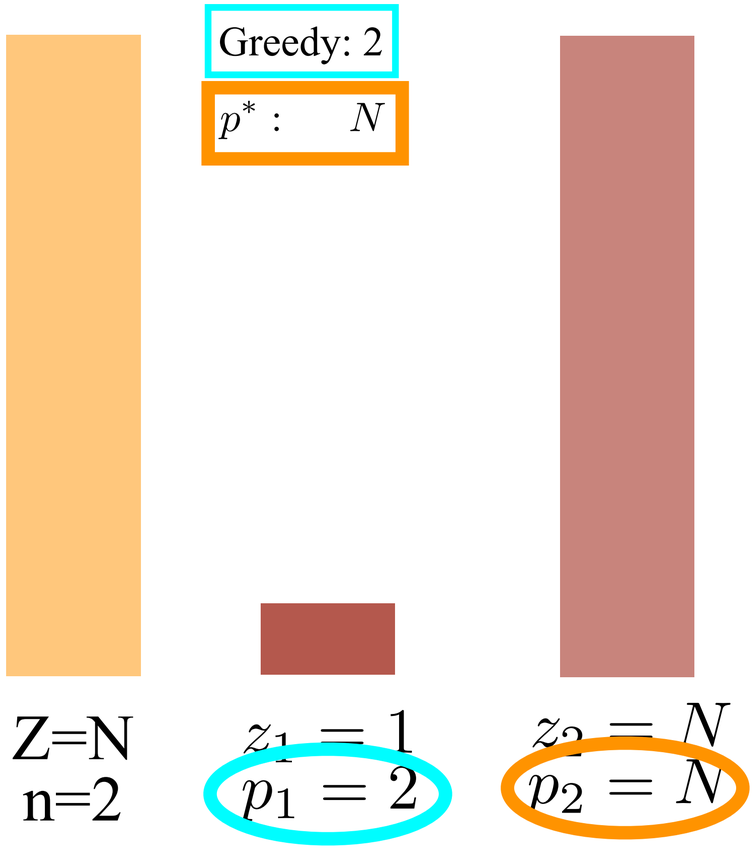
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Maximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$

Verbesserung?!



Algorithmus 4.3 GREEDY$_0$'

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ und G_0

mit

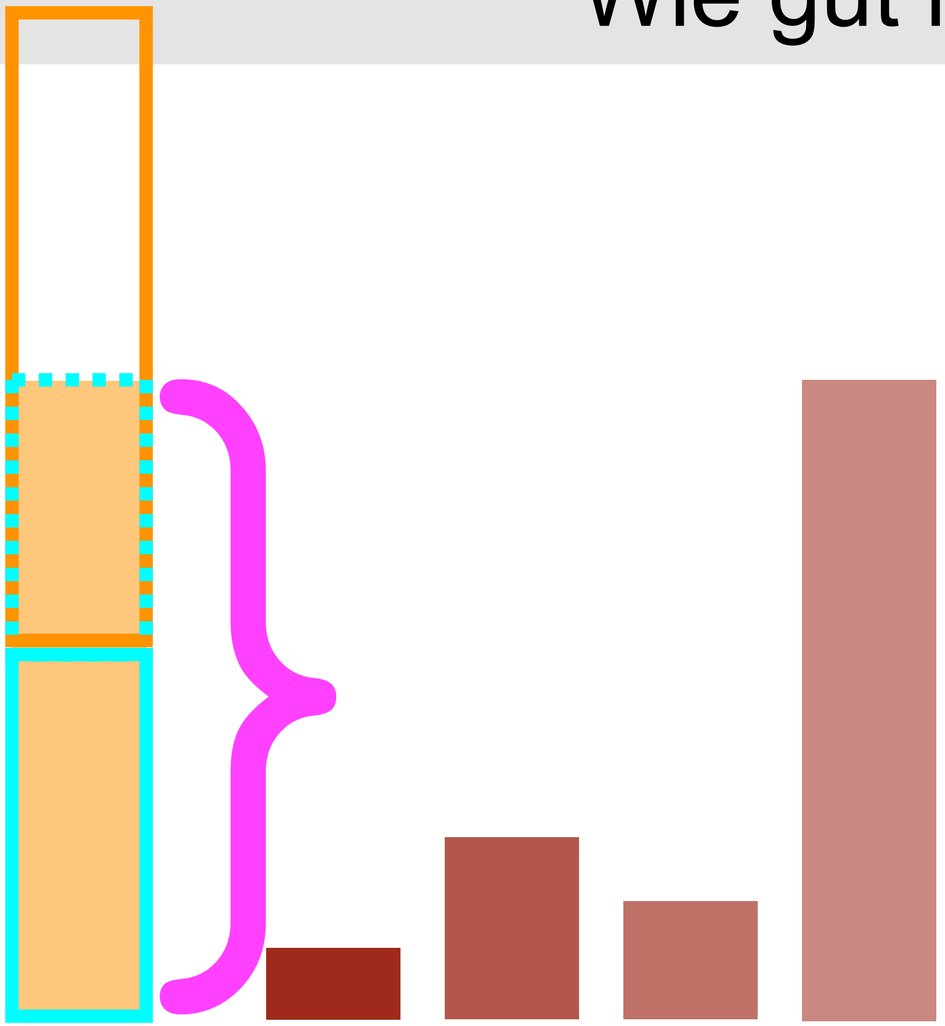
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = Maximal$$

1: Sortiere {1,...,n} nach $\frac{z_i}{p_i}$ aufsteigend;
Dies ergibt die Permutation $\pi(1), ..., \pi(n)$.
Setze $j = 1$.
2: **while** $(j \leq n)$ **do**
3: **if** $\left(\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z\right)$ **then**
4: $x_{\pi(j)} := 1$
5: $j := j + 1$
6: $G_0' := max\{G_0, max\{p_i \mid z_i < Z, i \in \{1, ..., n\}\}\}$
7: **return**

Wie gut ist das allgemein?



Algorithmus 4.3 Greedy-Approximation

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$

Ausgabe: $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G_0' := \sum_{i \in S} p_i$

1: $G_0' := max\{G_0, max\{p_i \mid z_i < Z, i \in \{1, ..., n\}\}\}$

2: **return**

Satz 4.4. *Algorithmus 4.3 berechnet eine Lösung mit G_0'; im Vergleich mit dem Optimalwert OPT gilt: $G_0' \geq \frac{1}{2} OPT$.*

$$G_0' \geq G_0 \qquad G_0' \geq p^*$$

$$OPT \leq FractionalKP \leq G_0 + x_i p_i \leq G_0 + p^* \leq 2G_0'$$

4.3 Approximationsalgorithmen

Approximation

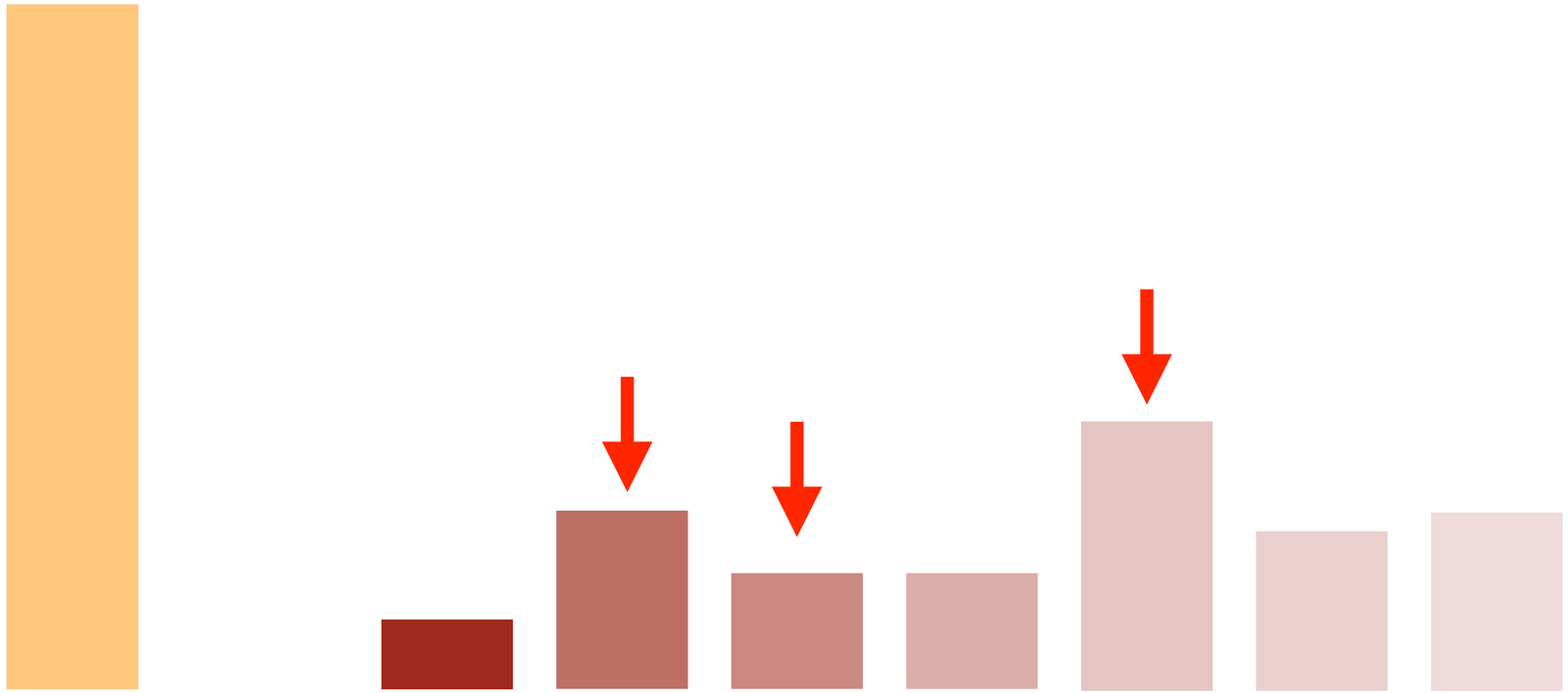
Definition 4.5 (Approximationsalgorithmus)**.**

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz I von MAX
   a) ALG in *polynomieller Zeit* in der Größe von I eine zulässige Lösung mit Wert ALG liefert
   b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$ (dabei ist $c \leq 1$)
2. Entsprechend für Minimierungsproblem wiederum:
   a) ALG liefert in *polynomieller Zeit* in der Größe von I eine zulässige Lösung mit Wert $\text{ALG}(I)$
   b) es gilt für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$: $\text{ALG}(I) \leq c \cdot \text{OPT}(I)$ (dabei ist $c \geq 1$)

4.4 Bessere Approximation

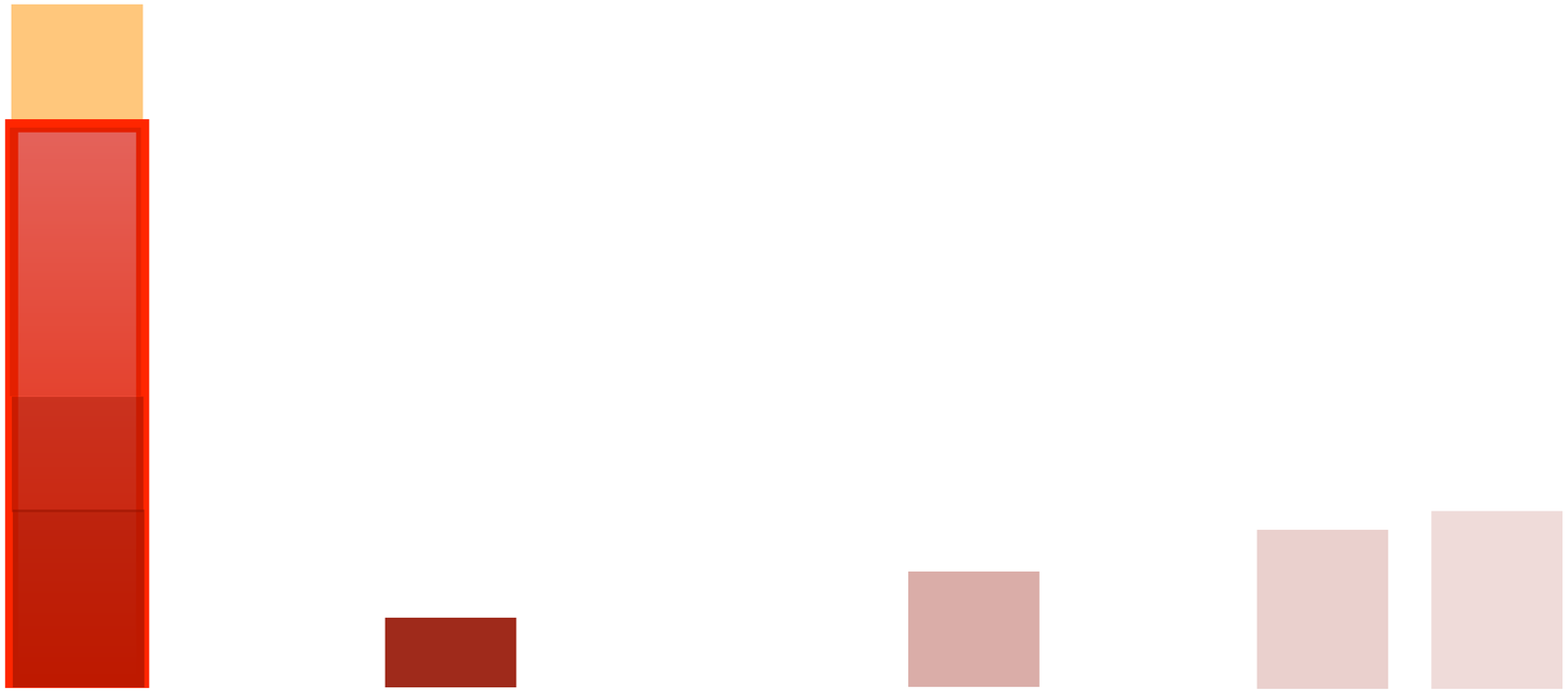
Ideen:

- Verwende die *k* wertvollsten Objekte aus einer Optimallösung.



Ideen:

- Verwende die *k* wertvollsten Objekte aus einer Optimallösung.



Ideen:

- Verwende die *k* wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.

Ideen:

- Verwende die *k* wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Ideen:

- Verwende die *k* wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Wie findet man die *k* wertvollsten Objekte?

- Enumeriere alle Teilmengen der Größe k! $\rightarrow O(n^k)$ Teilmengen

Bessere Approximation von Knapsack

Algorithmus 4.6 GREEDY$_k$

Eingabe: $z_1, ..., z_n, Z, p_1, ..., p_n$ [Parameter k fixiert]

Ausgabe: $S \subseteq \{1, ..., n\}$ mit $\sum_{i \in S} z_i \leq Z$ und Wert $G_k := \sum_{i \in S} p_i$

1: $G_k := 0$, $S := \emptyset$

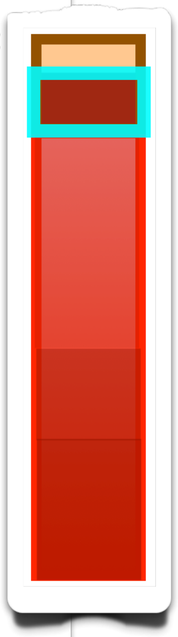
2: **for all** $\bar{S} \subseteq \{1, ..., n\}$ mit $|\bar{S}| \leq k$ **do**

3: **if** $(\sum_{i \in \bar{S}} z_i \leq Z)$ **then**

4: $G_k := max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i \{p_i | i \notin \bar{S}\})\}$;

5: Update S;

6: **return** G_k, S



Gütegarantie

Satz 4.7. GREEDY$_k$ *ist ein* $(1 - \frac{1}{k+1})$*-Approximationsalgorithmus.*

Definition 4.5 (Approximationsalgorithmus)**.**

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz I von MAX

   a) ALG in *polynomieller Zeit* in der Größe von I eine zulässige Lösung mit Wert ALG liefert

   b) für den Vergleich mit dem zugehörigen Optimalwert $\text{OPT}(I)$ gilt: $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$ (dabei ist $c \leq 1$)

Nicht 2^n, sondern n^2

(a) Die Laufzeit ist nicht mehr als $O(n^{k+2})$ - also polynomiell.

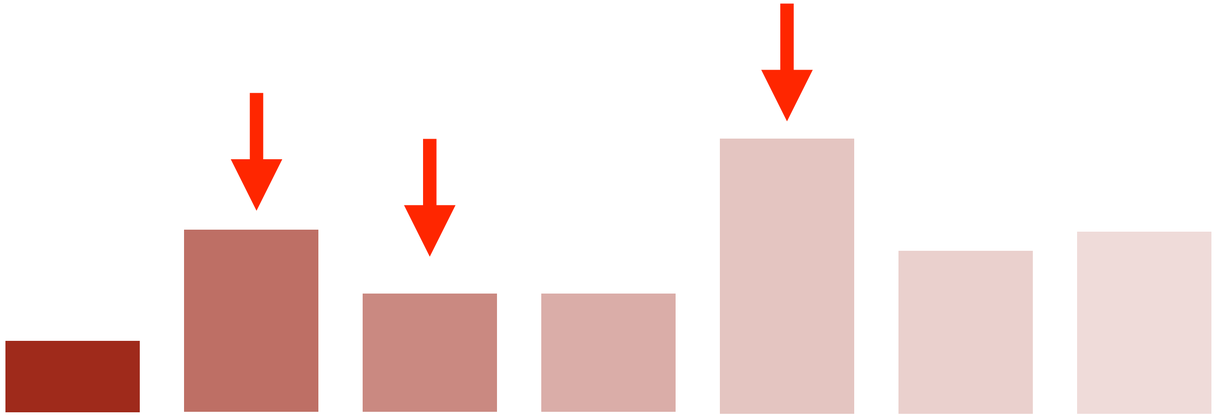
Gütegarantie

Satz 4.7. $\textsc{Greedy}_k$ *ist ein* $(1 - \frac{1}{k+1})$*-Approximationsalgorithmus.*

b) für den Vergleich mit dem zugehörigen Optimalwert $\textsc{opt}(I)$ gilt: $\textsc{alg}(I) \geq c \cdot \textsc{opt}(I)$ (dabei ist $c \leq 1$)

Betrachte OPT mit $S^* \subseteq \{1, \ldots, n\}$.

$|S^*| \leq k \implies$



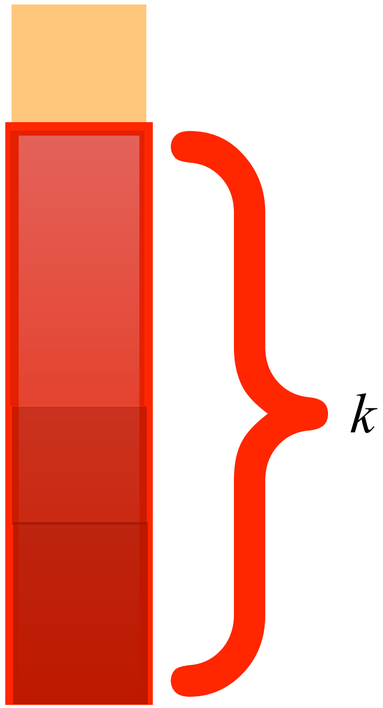
Gütegarantie

Satz 4.7. $\textsc{Greedy}_k$ *ist ein* $(1 - \frac{1}{k+1})$*-Approximationsalgorithmus.*

b) für den Vergleich mit dem zugehörigen Optimalwert $\textsc{opt}(I)$ gilt: $\textsc{alg}(I) \geq c \cdot \textsc{opt}(I)$ (dabei ist $c \leq 1$)

Betrachte OPT mit $S^* \subseteq \{1, \ldots, n\}$.

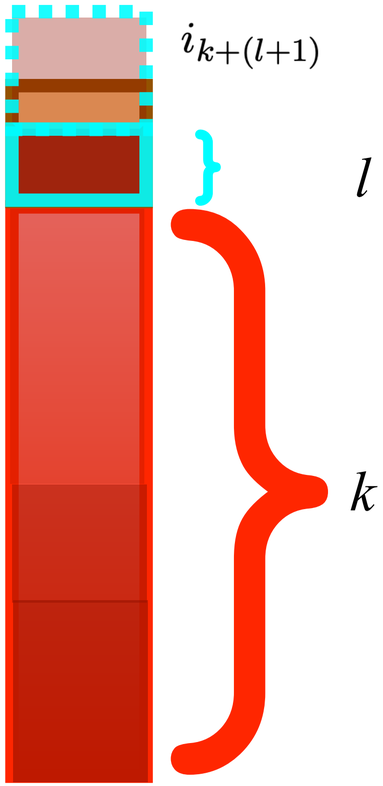
$|S^*| \leq k \implies$ OPT wird gefunden.



Gütegarantie

Satz 4.7. $\textsc{Greedy}_k$ *ist ein* $(1 - \frac{1}{k+1})$*-Approximationsalgorithmus.*

b) für den Vergleich mit dem zugehörigen Optimalwert $\textsc{opt}(I)$ gilt: $\textsc{alg}(I) \geq c \cdot \textsc{opt}(I)$ (dabei ist $c \leq 1$)



Betrachte OPT mit $S^* \subseteq \{1, \ldots, n\}$.

$|S^*| > k \implies$

a) $\sum_{j=1}^{k} p_{i_j} + \sum_{j=1}^{l} p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq OPT$

b) $\sum_{j=1}^{k} p_{i_j} + \sum_{j=1}^{l} p_{i_{k+j}} \leq G_k$

c) $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$$G_k + \frac{1}{k} G_k \geq \text{OPT}$$

$$\Leftrightarrow G_k \geq \frac{k}{k+1} \text{OPT}$$

$$= \left(1 - \frac{1}{k+1}\right) \text{OPT}$$

Zusatzbemerkungen

Man kann zeigen:

- Für $\textsc{Greedy}_k$ ist der Faktor $(1 - \frac{1}{k+1})$ bestmöglich.
- Wenn man G_0 in Zeile 4 von Algorithmus 4.6 durch G_0' ersetzt, bekommt man einen Approximationsalgorithmus mit Gütegarantie $(1 - \frac{1}{k+2})$.

Approximationschemata

Man sieht: Für jedes feste $\epsilon > 0$ gibt es einen polynomiellen Algorithmus für Knapsack, der eine $(1 - \epsilon)$-Approximation liefert. Das motiviert:

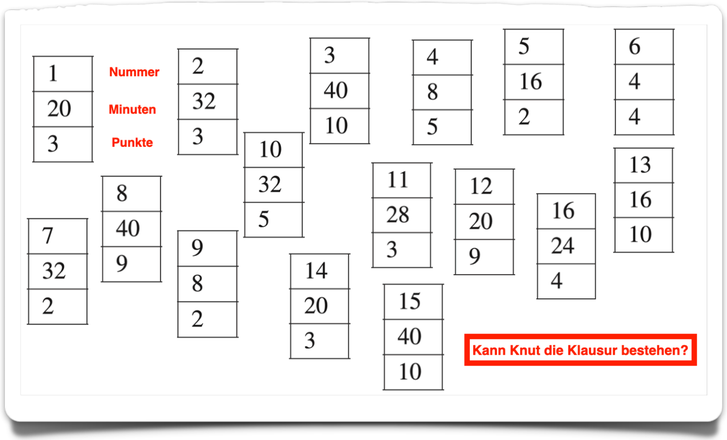
Definition 4.8 (PTAS). Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste $\epsilon > 0$ einen $(1 - \epsilon)$-Approximationsalgorithmus (bzw. $(1 + \epsilon)$) liefert.

Korollar 4.9. $\{\text{GREEDY}_k \mid k \in \mathbb{N}\}$ *ist ein PTAS für Knapsack.*

4.4 Ausblicke

Approximationsfaktoren

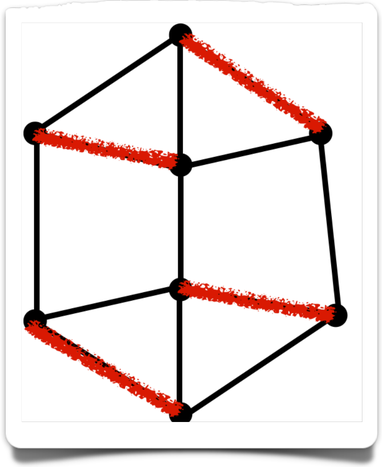
0-1 Knapsack



Max

$$1 - \frac{1}{k+2}$$

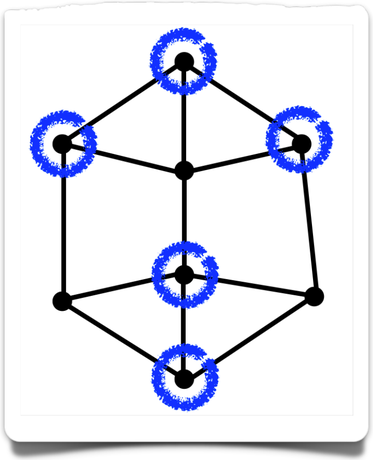
Matching



Max

1

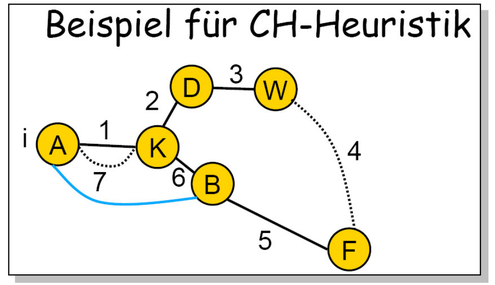
Vertex Cover

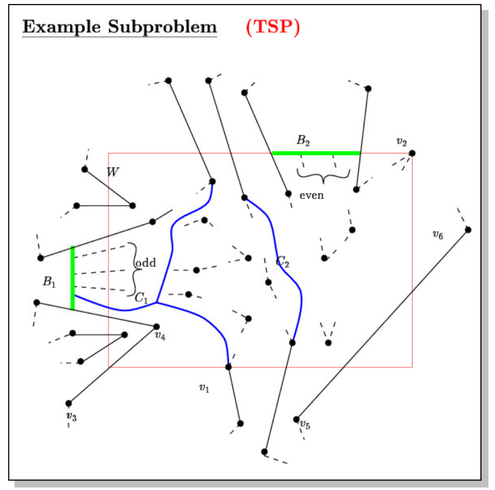


Min

2

Traveling Salesman





Min

$\frac{3}{2}$

$1 + \varepsilon$

Vielen Dank!

s.fekete@tu-bs.de